Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Engineering 38 (2012) 1905 – 1915

Procedia Engineering

www.elsevier.com/locate/procedia

Conference Title

Load Distribution Design Pattern for Genetic Algorithm Based Autonomic Systems

Vishnuvardhan Mannava[a] and T. Ramesh[b]

[a] Department of Computer Science and Engineering,
K L University, Vaddeswaram, 522502, A.P., India
vishnu@kluniversity.in
[b] Department of Computer Science and Engineering,
National Institute of Technology, Warangal, 506004, A.P., India
rmesht@nitw.ac.in

Abstract

The need for adaptability in software is growing, driven in part by the emergence of pervasive and autonomic computing. In many cases, it is desirable to enhance existing programs with adaptive behaviour, enabling them to execute effectively in dynamic environments. Increasingly, software systems should self-adapt to satisfy new requirements and environmental conditions that may arise after deployment. Due to their high complexity, adaptive programs are difficult to specify, design, verify, and validate. In this paper we propose a new approach for Genetic Algorithm based multi objective evolution in autonomic system using Design Patterns. Proposed system satisfies properties of autonomic system. Proposed system distributes the population of Genetic Algorithm to different clients to execute the population and store population results into database. We use different Design Patterns for this autonomic system those are Case Based Reasoning, Database Access Design Pattern and Master Slave. Main objective of the system is to reduce the load of the system to distribute the population. . The pattern is described using a java-like notation for the classes and interfaces. A simple UML class and Sequence diagrams are depicted.

Keywords: Design Patterns; Distributed System; Genetic Algorithms; Database Access Pattern; and Autonomic System.

doi:10.1016/j.proeng.2012.06.233

1. Introduction

A Genetic Algorithm (GA) is a problem solving method inspired by Darwin's theory of evolution: a problem is solved by an evolutionary process resulting in a best (fittest) solution (survivor). In a GA application, many individuals derive, independently and concurrently, competing solutions to a problem. These solutions are then evaluated for fitness and individuals survive and reproduce based upon their fitness. Eventually, the best solutions emerge after generations of evolution.

The flow of a typical GA simulation is as follows: First, a GA server creates many individuals randomly. Each of these individuals is tested for fitness. Based on their fitness, measured by a fitness function that quantifies the optimality of a solution, the server selects a percentage of the individuals that are allowed to crossover with each other, analogous to gene sharing through reproduction in biological organisms. The crossover between two parents produces offspring, which have a chance of being randomly mutated. A child thus produced is then placed into the population for the next generation, in which it will be evaluated for fitness. The process of selection, crossover, and mutation repeats until the new population is full and the new generation repeats the behaviour of the previous generation. After many generations, the individuals are expected to become more adept at solving the problem to which the GA is being applied.

In order for a GA simulation to work well, there needs to be a significant number of individuals within a population, and the simulation needs to be allowed to run for many generations. Furthermore, the simulation will typically need to be run repeatedly while parameters such as mutation rate, population size and crossover functions are tuned. Thus, a successful GA simulation requires the calculation of the fitness function thousands of times or more. It is therefore critical that the function that performs the calculation of the fitness, called the fitness function, can be executed as speedily as possible.

Design Patterns have, over the last decade, fundamentally changed the way we think about the design of large software systems. Using Design Patterns not only helps designers exploit the community's collective wisdom and experience as captured in the patterns, it also enables others studying the system in question to gain a deeper understanding of how the system is structured, and why it behaves in particular ways. And as the system evolves over time, the patterns used in its construction provide guidance on managing the evolution so that the system remains faithful to its original design, ensuring that the original parts and the modified parts interact as expected. Although they are not components in the standard sense of the word, patterns may, as has been noted, be the real key to reuse since they allow the reuse of design, rather than mere code. But to fully realize these benefits, we must ensure that the designers have a thorough understanding of the precise requirements their system must meet in applying a given pattern, as well as automated or semi-automated ways of checking whether the requirements have been satisfied. To that end, the work we present in describes an approach to specifying Design Patterns precisely using formal contracts. Our goal in this paper is to extend that work, and to develop a runtime monitoring approach that allows system designers to determine whether the patterns used in constructing a system have been applied correctly. We use an aspect-oriented programming approach to achieve this goal.

Distributed computing applications grow in size and complexity in response to increasing computational needs, it is increasingly difficult to build a system that satisfies all requirements and design constraints that it will encounter during its lifetime. Many of these systems must operate continuously, disallowing periods of downtime while humans modify code and fine-tune the system. For instance, several studies document the severe financial penalties incurred by companies when facing problems such as data loss and data inaccessibility. As a result, it is important for applications to be able to self-reconfigure in response to changing requirements and environmental conditions. IBM proposed autonomic computing as a means for automating software maintenance tasks. Autonomic computing refers to any system that manages itself based on a system administrator's high level objectives while incorporating capabilities such as self-reconfiguration and self-optimization. Typically, developers encode reconfiguration strategies at design time, and the reconfiguration tasks are influenced by

anticipated future execution conditions. We propose an approach for incorporating Genetic Algorithms as part of the decision-making process of an autonomic system. This approach enables a decision making process to dynamically evolve reconfiguration plans at run time.

Based on literature we propose a new pattern for Genetic Algorithm evaluation by using Design Pattern. Proposed pattern server evaluate Genetic Algorithm based on fitness function, server generates different population. Population evaluated by different client at a time, this will reduce server load. All results of clients are stored in database; Fig 1 will show the autonomic system [5].

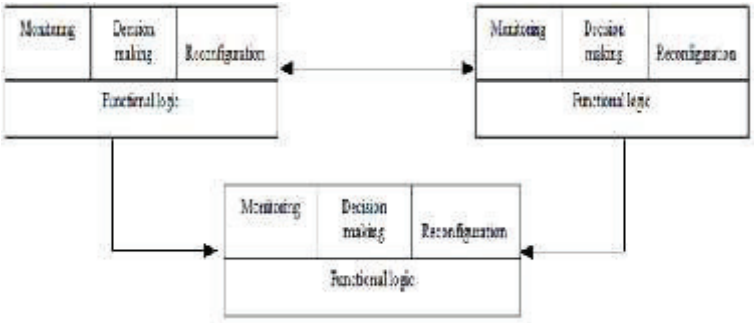

Fig 1: Autonomic System

Proposed system solves multi object optimization using Genetic Algorithms. Multi objective optimization is A vector of decision variables which satisfies constraints and optimizes a vector function whose elements represent the objective functions. These functions form a mathematical description of performance criteria which are usually in conflict with each other. Hence, the terms “optimize” means finding such a solution which would give the values of all the objective functions acceptable to the decision maker.

The decision variables are the numerical quantities for which values are to be chosen in an optimization problem. These quantities are denoted as x_j =1, 2,..., n. The vector x of n decision variables is represented by:

$$X = \begin{pmatrix} X_1 \\ X_2 \\ . \\ . \\ . \\ X_n \end{pmatrix}$$

This can be written more conveniently as: $x = [x_1, x_2, ..., x_n]^T$, where T indicates the transposition of the column vector to the row vector.

$g_i(x) \leq 0$ i =1,...,m or equalities:

$h_j(x) = 0$ j =1,...,p

Note that p, the number of equality constraints , must be less than n, the number of decision variables, because if $p \geq n$ the problem is said to be over constrained , since there are no degrees of freedom left for optimizing (i.e., in other words, there would be more unknowns than equations). The number of degrees of freedom is given by $n - p$. Also, constraints can be explicit (i.e., given in algebraic form) or implicit, in which case the algorithm to compute $g_i(x)$ for any given vector x must be known. Multi objective optimization problem is solved by using Genetic Algorithm. Proposed architecture solve multi objective

optimization using Genetic Algorithm, architecture run distributed Genetic Algorithm for finding components that are useful to compose a new component.

2. Related Work

In this section we present some works that deal with different aspects of autonomic systems and their design. Nick Burns and Mike Bradley paper[1] discuss applying Genetic Algorithm for distributing computing we take this paper is base paper here we are applying Genetic Algorithms and Design Patterns in autonomic systems. The author of the paper uses compostie, singleton half-sys and half-asyn patterns for system designing. In Jason O. Hallstrom and Neelam Soundarajan[2] uses observer pattern for monitoring approach for determining whether the pattern contracts used in developing a system are respected at runtime. In Andres J. Ramirez and David B. Knoester [3] proposes applying Genetic Algorithms for decision making in autonomic computing.

In Andres J. Ramirez and David B. Knoester [4] uses Distributed Adapters Pattern (DAP) in the context of remote communication between two components for object oriented applications. In this paper uses all base paper concepts for designing new system for autonomic computing. Genetic Algorithms also have been used to design overlay multicast networks for data distribution [9]. These overlay networks must balance the competing goals of diffusing data across the network as efficiently as possible while minimizing expenses. A common approach for integrating various objectives in a Genetic Algorithm is to use a cost function that linearly combines several objectives as a weighted sum [12]. Although most of these approaches [10] achieved rapid convergence rates while producing overlay networks that satisfied the given constraints, to our knowledge, the methods were not applied at run time to address dynamic changes in the network's environment [11].

3. Load Distribution Design Pattern Template

To facilitate the organization, understanding, and application of the adaptation Design Patterns, this paper uses a template similar in style to that used by Ramirez et al. [2]. Likewise, the Implementation and Sample Code fields are too application-specific for the Design Patterns presented in this paper.

3.1. Pattern Name:

Load Distribution Design Pattern.

3.2. Classification:

Structural – Monitoring-Decision Making

3.3. Intent:

Load Distribution Design Pattern main objective is to distribute load of genetic server to different clients. Generally every problem consists more than one solution in Genetic Algorithm server solve all possible solutions. Out of all solutions server will provide best solution as the result. So Genetic Algorithm server load will increase gradually our pattern distribute Genetic Algorithm population to different clients.

3.4. Motivation:

Main motive of Load Distribution Design Pattern is to distribute Genetic Algorithm population to different client for solving problem. Our goal is to reduce server load by distributing Genetic Algorithm population.

3.5. Proposed Pattern Structure:

A UML class diagram for the Load Distribution Design Pattern show in Fig 2.

Three Design Patterns are used for Load Distribution Design Pattern those are Case Based Reasoning, Database Access and Master Slave Design Patterns. Input class will request the server for solving multi object optimization problem server will pick appropriate fitness function for the problem by using case based reasoning Design Pattern. Based on fitness function server will generate population, population of server is distributed to different clients by using Master Slave Design Pattern. Results of different client are stored in database by using database access Design Pattern.

3.6. Participants:

(a) *Client:* Input Stream supplies Multi object Optimization problem to server, based on input stream problem server choose appropriate fitness function for population generation.
(b) *Server:* Server will take input from the input stream, based on the input it will find the fitness function with the help of the Case Based Reasoning Design Pattern, based on the fitness function server will find the possible chromosomes for the problem. Each chromosome is distributed different clients. After completion of the evaluation server collect results from server. Based on results server chooses appropriate solution as a result of the problem.
(c) *Client repository:* Client repository stores services that are invoked by client, repository will create separate thread for each client, if service is finished with in time stamp then it will report result to client otherwise it report service class, service class will take decision based on time stamp availability if time is available then it choose resume service otherwise it choose suspend service.
(d) *Decision:* This class represents a reconfiguration plan that will yield the desired behaviour in the system.
(e) *Fixed Rules:* This class contains a collection of Rules that guide the Inference Engineering producing a Decision. The individual Rules stored within the Fixed Rule scan be changed at run time.
(f) *Learner:* This is an optional feature of the Case based Reasoning Design Pattern.
(g) *Log:* This class is responsible for recording which reconfiguration plans have been selected during execution. Each entry is of the form Trigger-Rule-Decision.
(h) *Rule:* A Rule evaluates to true if an incoming Trigger matches the Trigger contained in the Rule.

3.7. Applicability:

Use the autonomic system using Design Pattern when

- The strategy chooses the reconfiguration plan based on the input stream of the modules.
- You need different variants of an algorithm. For example, you might define algorithms reflecting different space/time trade-offs. Strategies can be used when these variants are implemented as a class hierarchy of algorithms.

- If the strategy will store different fitness functions updating the fitness function are also possible in strategy.
- It reduces the workload of server; the system is suitable for distributed computing.

3.8. Related Design Patterns:

The intent of the Load Distribution Design Pattern is similar to the Configuration pattern. The Configuration pattern decouples structural issues related to configuring services in distributed applications from the execution of the services themselves. The Configuration pattern [1] has been used in frameworks for configuring distributed systems to support the construction of a distributed system from a set of components. In a similar way, the Service Administration Design Pattern decouples service initialization from service processing. The primary difference is that the Configuration pattern focuses more on the active composition of a chain of related services, whereas the Service Administration Design Pattern focuses on the dynamic initialization of service handlers at a particular endpoint. In addition, the Service Administration Design Pattern focuses on decoupling service behavior from the service's concurrency strategies.

The Manager Pattern [7] manages a collection of objects by assuming responsibility for creating and deleting these objects. In addition, it provides an interface to allow clients access to the objects it manages. The Service Administration Design Pattern can use the Manager pattern to create and delete Services as needed, as well as to maintain a repository of the Services it creates using the Manager Pattern. However, the functionality of dynamically configuring, initializing, suspending, resuming, and terminating a Service created using the Manager Pattern must be added to fully implement the Service Administration Pattern.

3.9. Roles of Our Design Patterns:

(a) *Case Based Reasoning:* Strategy Design Pattern will choose the fitness function based on the input stream. Fitness function will wary based on the input stream. It will choose appropriate fitness function that is suitable for Autonomic System. Strategy Design Pattern will use for decision making for Autonomic System.

(b) *Observer:* Observer Design Pattern will use for monitoring in Autonomic Systems. Observer will monitor the clients' responses and note the observations. Monitoring helps to reduce the server time (waiting for the client response). If a client is unable to produce result then Observer will inform the details of the client to the server.

(c) *Master Slave:* Singletons Design Pattern will use execute the population in client. The main objective of the Master Slave in this Autonomic System is each client will take only one population for execution.

(d) *Thread per Connection:* model, is applied to the interaction between the clients and the server. For each client, the server spawns a separate Server Thread dedicated to interacting with the client.

3.10. Interface Definition for Pattern entities:

Input Class:

```
Public class Inputclass
{
    public String Read(){….}
}
```

Server:

```
Public class Server
{
    Public string fitnessfunction()
    {
    i.concereteImpl();
    }
    Public int enqueuejob(int ){…}
    Public int jobqueue(int){….}
    Public int result(){…}
    Public getjob(){…}
}
```

Client:

```
Public class Client
{
    Public int dojob(){….}
}
```

Insersion:

```
 Public class Viewone
{
    Public int insert(object e){…}
}
```

Fixed rules:

```
Public class Fixedrules
{
    Public void Elaborate(String str) {…}
}
Public class rules implements Fixedrules
{
    Public void Elaborate(String str){….}
 }
```

Decision:

```
Public class Decision
{
    Public int action(object ){….}
}
```

Client repository:

```
class Clientrepository
 {
    private static Client instance = null;
    public static instance()
    {
      if( instance == null )
      {
        instance = new Singleton();
      }
      return instance;
  }}
```

4. Case Study

To demonstrate the efficiency of the pattern we took the profiling values using the Net beans IDE and plotted a graph that shows the profiling statistics when the pattern is applied and when pattern is not applied. This is shown in Fig 4.Here X-axis represents the runs and Y-axis represents the time intervals in milliseconds. Below simulation shows the graphs based on the performance of the system if the pattern is applied then the system performance is high as compared to the pattern is not applied.

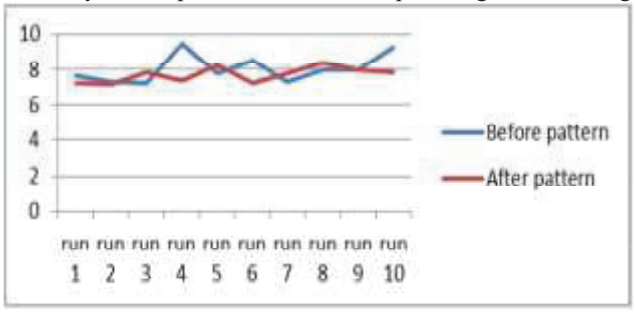

Fig 4: Profiling statistics before applying pattern and after applying pattern

The view of our proposed Design Pattern can be seen in the form of a Class Diagram see Figure 2.

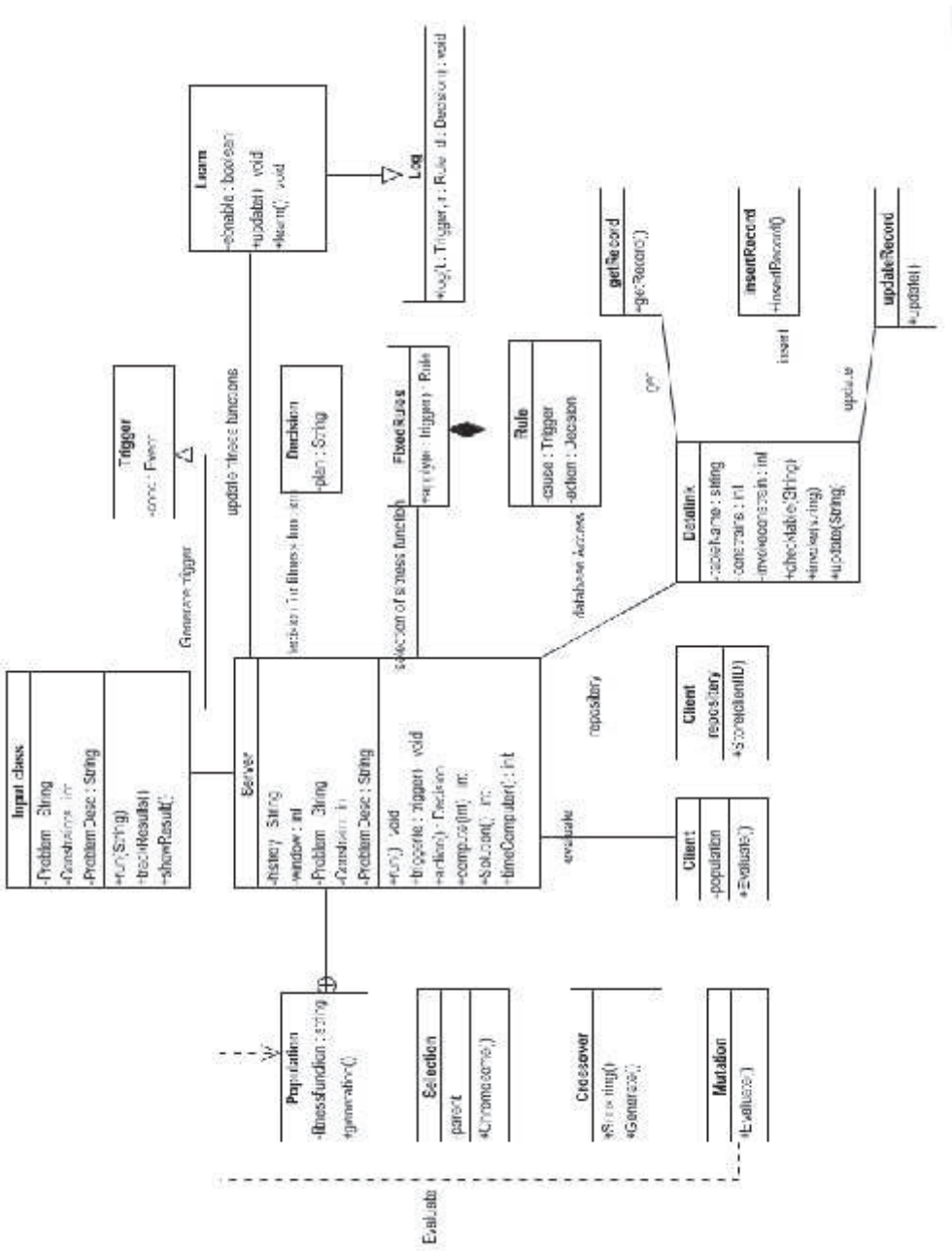

Fig 2: Class Diagram for Load Distribution Design Pattern.

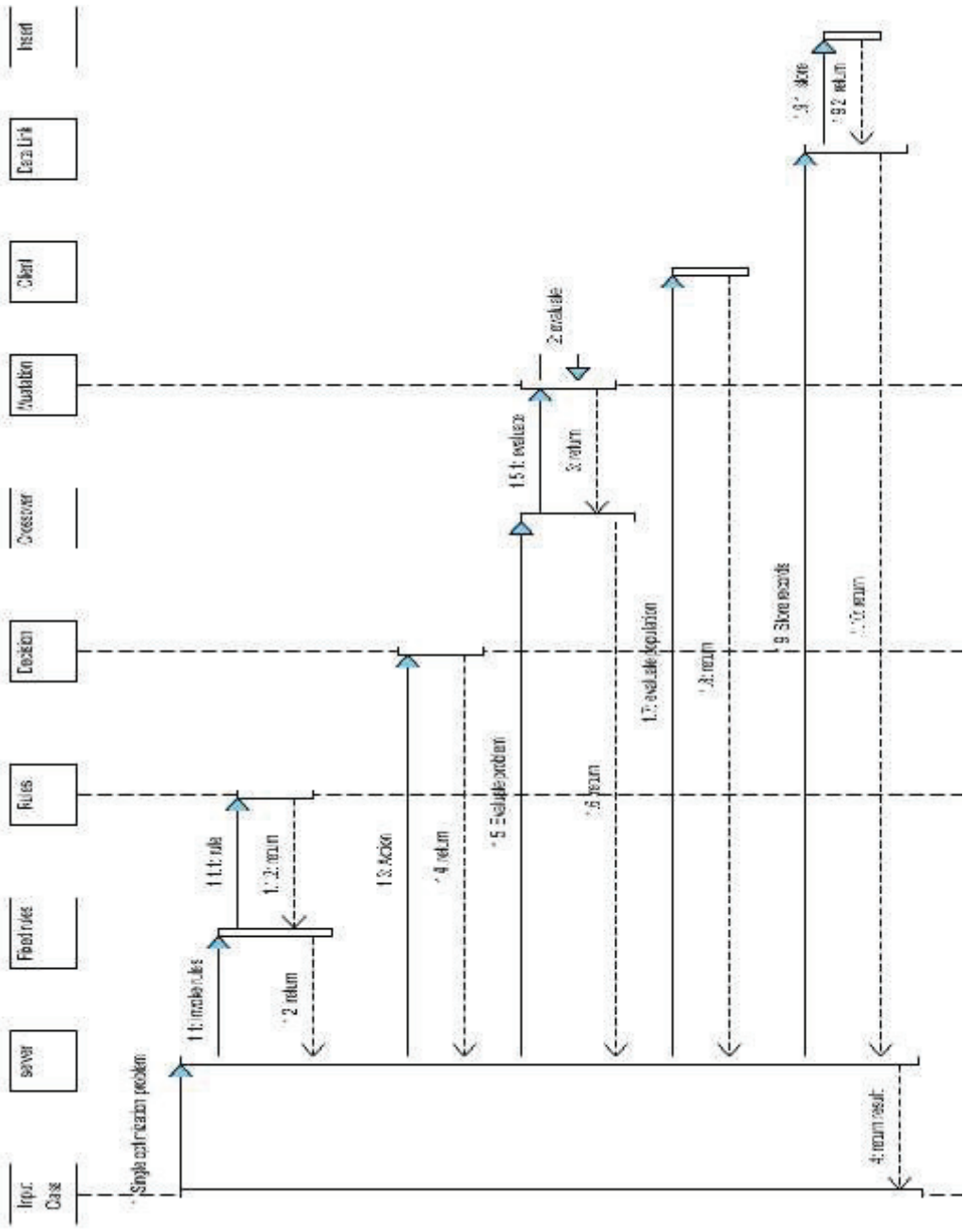

Fig 3: Sequence Diagram for Load Distribution Design Pattern.

5. Conclusion and Future Work

In this paper we prepare an approach for autonomic computing using Design Pattern. The system will satisfy all the features of the autonomic system except reconfiguration. The system will reduce workload of server by distributing the population to different clients. This paper implies four Design Patterns those are Case Based Reasoning, Database Access Patterns and Master Slave Design Pattern. Database Access Pattern is used for storing results in database. Based on the fitness function chromosomes assign to the different clients, clients will do the job and return the result of the job finally we will evaluate results and provide best result out of all possible results.

Future work: Future aims to develop a system that will distribute the Genetic Algorithm to different clients, and design autonomic system it will reconfigure based on autonomic changes in the system using Design Patterns.

6. Reference

[1] Nick Burns Mike Bradley and Mei-Ling L. Liu. Applying Design Patterns in Distributing a Genetic Algorithm Application, Proceedings of the International Conference on Software Engineering Research and Practice, SERP 2005, Las Vegas, Nevada, USA, Volume 1 2005; June 2005, doi:10.1.1.86.1708 .

[2] Andres J. Ramirez, David B. Knoester, Betty H.C. Cheng and Philip K. McKinley. Applying Genetic Algorithms to Decision Making in Autonomic Computing Systems, 6th International Conference in Autonomic Computing, ICAP, ACM ; 2009

[3] Jason O. Hallstrom, Neelam Soundarajan and Benjamin Tyler. Monitoring Design Pattern Contracts, 8th International Conference on Software Engineering Knowledge Engineering, SEKE, San Francisco; 2006, doi:10.1.1.130.4843.

[4] Vander Alves *and* Paulo Borba. Distributed Adapters Pattern: A Design Pattern for Object-Oriented Distributed, ACM Digital Library; 2005, doi: 10.1.1.20.4519.

[5] Kent Beck, James Coplien, Ron Crocker, Lutz Dominick et.al. Industrial Experience with Design Patterns, ICSE; 1996, doi: 10.1109/ICSE.1996.493406 .

[6] V.S.Prasad Vasireddy, Vishnuvardhan Mannava, and T.Ramesh. A Novel Event Based Autonomic Design Pattern for Management of Web Services, Communications in Computer and Information Science, 1, Volume 198, Advances in Computing and Information , Springer-Verlag Berlin Heidelberg; 2010, Pages 142-151, doi:10.1007/978-3-642-22555-0.

[7] Vishnuvardhan mannava and T.Ramesh. A Novel Adaptive Monitoring Compliance Design Pattern for Autonomic Computing Systems, Communications in Computer and Information Science, Advances in Computing and Communications, Part 3, Springer-Verlag Berlin Heidelberg; 2010, Pages 250-259, doi:10.1007/978-3-642-22709-7_26.

[8] Ramirez, A.J. Design Patterns for Developing Dynamically Adaptive Systems. Master's thesis, Michigan State University, East Lansing, Michigan; 2008, doi: 10.1145/1808984.1808990.

[9] W. Pree. Design Patterns for Object-Oriented Software Development. Reading, MA: Addison-Wesley; 1994.

[10] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Design Patterns. Addison-Wesley, Reading, Mass; 1994.

[11] S.Crane, J.Magee, N. Pryce. Design Patterns for Binding in Distributed Systems, The OOPSLA '95 Workshop on Design Patterns for Concurrent, Parallel, and Distributed Object-Oriented Systems, (Austin, TX), ACM; 1995, doi:10.1.1.39.7601.

[12] Shang-Wen Cheng, David Garlan, Bradley Schmer. Architecture-based self adaptation in the presence of multiple objectives, International workshop on Self-adaptation and Self-managing systems, New York, ACM; 2006, doi: 10.1145/1137677.1137679.